UNIVERSIDADE ESTADUAL DE PONTA GROSSA

SETOR DE ENGENHARIAS, CIÊNCIAS AGRÁRIAS E DE TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

DAVID RAFAEL CARRIEL DE LIMA

DILUAN LIMA DE MATOS

GABRIEL KUTNER NOVOTNI

HENDRIK GUARNERI

THIAGO FELIPE DE FREITAS

PADRÕES DE PROJETO

ENGENHARIA DE SOFTWARE

PONTA GROSSA

2023

DAVID RAFAEL CARRIEL DE LIMA

DILUAN LIMA DE MATOS

GABRIEL KUTNER NOVOTNI

HENDRIK GUARNERI

THIAGO FELIPE DE FREITAS

PADRÕES DE PROJETO

Trabalho para ser entregue na disciplina

“Engenharia de Software” ministrada pelo

professor Rafael Fernandes Siqueira;

Acadêmicos: David Rafael Carriel de Lima,

Diluan Lima de Matos, Gabriel Kutner

Novtni, Hendrik Guarneri e Thiago Felipe de Freitas;

Ra’s: 21007423, 21007623, 21001623,

21002223;

Turma: NA

PONTA GROSSA

2023

**SUMÁRIO**

**PADRÕES DE PROJETO**

Desenvolver um projeto de software é uma tarefa bem complexa, e que pode ser bem difícil dependendo de sua magnitude, porém, com experiência e prática, um padrão fica evidente: Vários problemas recorrentes se resolvem da mesma maneira, mesmo que estejam em projetos totalmente diferentes, e é isso que pode ser definido como um “padrão de projeto”.

Por definição, são soluções para erros e problemas recorrentes na engenharia de software, mas uma coisa precisa ser ressaltada, padrões de projeto não são algoritmos. Um algoritmo age de forma direta, definindo instruções passo-a-passo, já um padrão de projeto age em um nível de design, ou seja, ele atua em um alto nível de programação, fornecendo uma estrutura flexível para seu projeto.

Para nortear esses padrões, eles são divididos em 3 tipos: Criacional, Estrutural e Comportamental, suas definições serão abordadas ao longo desse trabalho.

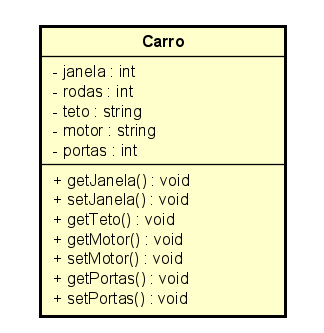
**PADRÕES CRIACIONAIS – BUILDER**

O objetivo desse tipo de padrão é fornecer novos mecanismos para a criação de objetos, que é o que dá nome ao modelo, esses objetos fornecem uma maior flexibilidade no projeto, reutilizando o código que já foi criado.

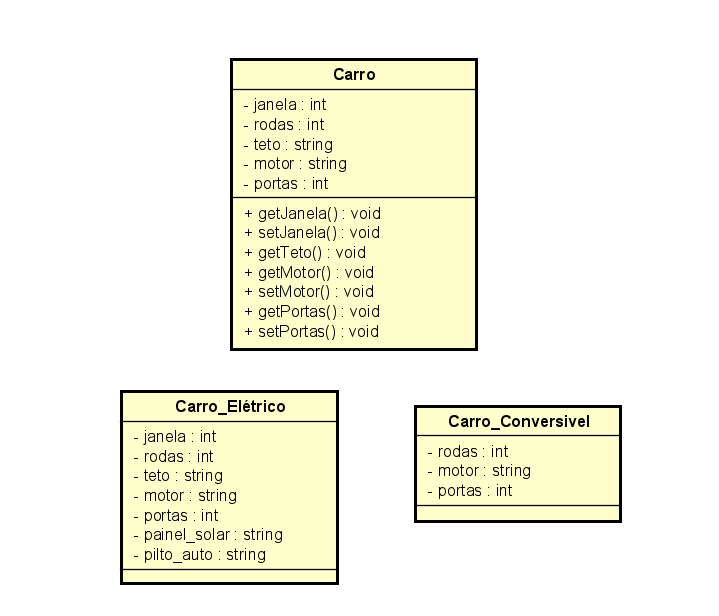
Uma observação deve ser feita antes de prosseguir, padrões de projeto são mais focados na POO (programação orientada a objetos), portanto, é comum haver, além do código, uma representação do problema em UML de suas classes.

* Problema: Imagine que você tenha um objeto bem complexo, que para ser iniciado, precisa de muitos parâmetros, isso gera um construtor de classes com muito código, o que foge um pouco da proposta do construtor, que é de ser eficiente para criar suas classes. Existe uma possível solução para isso, mas que também é meio problemática: A criação de subclasses, mas para ficar mais claro, seguiremos com um exemplo:

Imagine que, uma parte da implementação de seu projeto, gire em torno da criação de carros, todo carro tem alguns elementos fundamentais, como o motor, as rodas, os vidros, etc; Porém, há elementos únicos de determinados tipos de carros, há carros que tem 5 assentos, enquanto outros tem apenas 2, incluir esse tipo de especificação vai fazer o seu código ser extremamente longo sem necessidade. Também não é muito viável criar subclasses com cada tipo de carro, o código continua muito extenso, e é aí que entra o Builder.

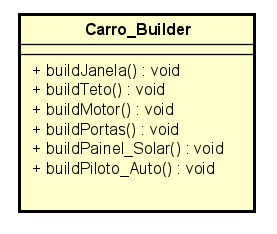


Primeiro Caso:



Segundo Caso:

* Solução: O builder possibilita a criação de objetos em etapas, sem a necessidade de especificar todos os atributos de uma vez, seguindo pelo exemplo: O construtor de classes “Carro” terá métodos como: colocar\_janelas, colocar\_teto, colocar\_portas, assim, é possível chamar apenas aquilo que for necessário para a construção do carro em específico, um carro conversível, por exemplo, não chamará o método colocar\_teto.



Com Builder:

Concluindo: O builder é um padrão criacional capaz de separar o código de criação do código de utilização, permitindo fazer um objeto em etapas, sem sobrecarregar o código, e reutilizando de maneira organizada aquilo que já foi programado. Sua utilização é usada em objetos complexos, com muitos parâmetros, e esse acaba justamente sendo sua “desvantagem”, ele pode ser considerado um dos padrões mais complexos de se entender e aplicar.

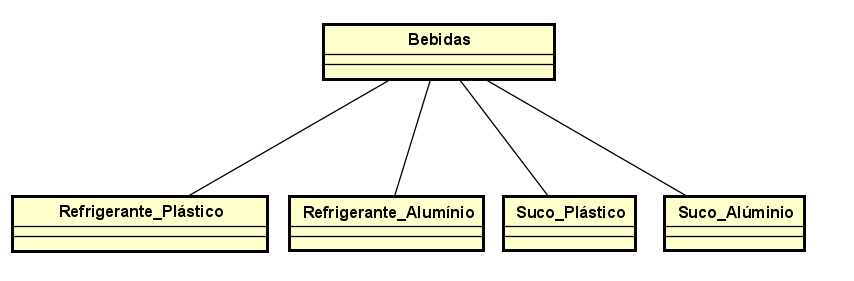
**PADRÕES ESTRUTURAIS – BRIDGE**

Os padrões estruturais, diferente dos criacionais, não tem seu foco na criação de objetos, e sim na criação de estruturas, que reúnem objetos e classes, mantendo também sua flexibilidade e eficiência, condensando o conteúdo para que hajam menos objetos soltos e maiores estruturas.

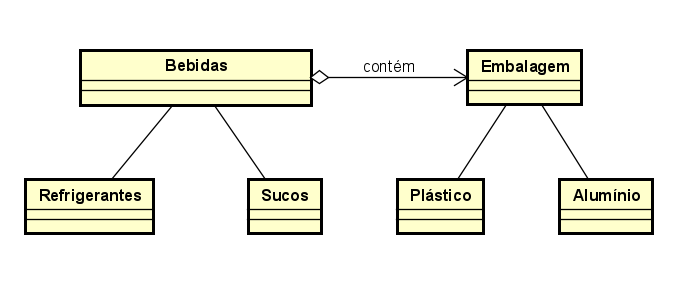
As palavras-chave desse padrão são: Abstração e Implementação, a abstração é um código de alto nível que normalmente define ações para algum objeto, já a implementação é o código que vai realmente fazer o trabalho. O problema desse modelo surge justamente nesses momentos do projeto.

* Problema: Enquanto você faz a abstração de uma nova subclasse no seu código, ela pode acabar entrando em conflito com outras subclasses na hora de sua implementação. O exemplo utilizado agora será o da criação de bebidas: Imagine que você criou uma classe intitulada “Bebida”, e ela pode ter as seguintes embalagens (subclasses): Plástico ou Alumínio. Então, você decide adicionar mais variedade a essa criação de bebidas, colocando as seguintes subclasses: Refrigerantes e Sucos.

Isso vai criar uma confusão bem grande na sua estrutura, pois você acaba misturando elementos que não tem tanta conexão entre si, e deixa o código com muitas possibilidades, como pode ser visto na imagem a seguir:



* Solução: Assim como o nome sugere, a solução é fazer uma “ponte” entre as subclasses, esse tipo de problema surge quando a abstração é feita de maneira confusa, misturando elementos que não precisam se ligar diretamente. Nesse exemplo, seria criado uma nova classe chamada “Embalagem”, e a hierarquia seria alterada, para que a “bebida” contenha “embalagem”.

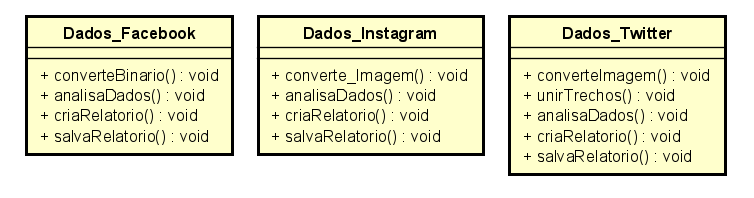


Concluindo: Bridge é um padrão mais simples de se entender, e com uma solução igualmente simples, a ideia por trás desse modelo é trazer mais estabilidade ao seu código, fazer com que ele esteja simples eventuais conexões ou alterações, a sua única desvantagem é quando esse padrão se aplica de forma errônea, separando classes que poderiam estar juntas de forma coesa, o que pode também sobrecarregar o código.

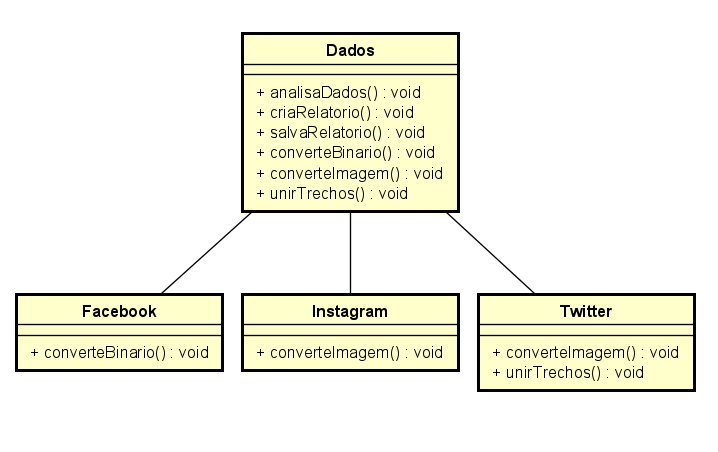
**PADRÕES COMPORTAMENTAIS – TEMPLATE METHOD**

Diferente dos demais padrões, os comportamentais não agem especificamente em um objeto ou estrutura, mas sim entre eles, definindo conexões e comunicações, delegando responsabilidades para os elementos.

* Problema: Imagine que você tem dois objetos distintos, porém, com várias semelhanças, e que por algum motivo externo, não podem estar na mesma classe ou reutilizar interfaces/implementações. Exemplificando: Imagine que você está programando um software que analisa um perfil em diferentes redes sociais e cria um relatório com suas informações, por mais que o código de gerar os relatórios e armazenar eles seja o mesmo, cada rede social salva seus dados em tipos diferentes de arquivos, o que ocasiona uma grande repetição de código, exigindo uma classe para cada rede social.

Exemplo de repetição de métodos em diferentes classes:

* Solução: O Template Method surge com a função de “fundir” essas classes, em uma superclasse, e aí fazer novas chamadas apenas para aqueles métodos que precisam ser substituídos. No caso do exemplo, será criada uma superclasse chamada “Dados”, com subclasses que chamam apenas os métodos exclusivos daquela rede social.

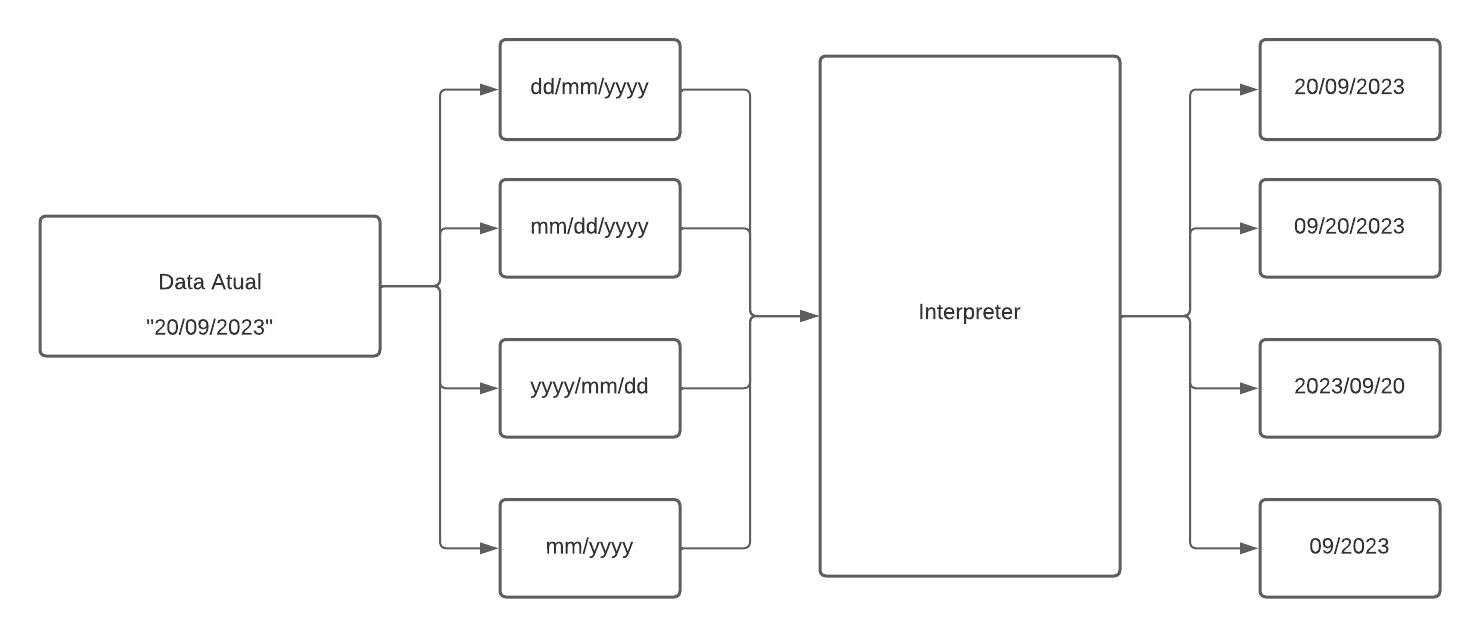
Exemplo com Template Method

Concluindo: Esse padrão pode gerar alguns problemas quando mal implementado, além da facilidade de erros de abstração surgirem, porém, é possível reduzir e flexibilizar o código dessa forma, sendo muito útil para processos que se repetem com uma grande frequência.

**INTERPRETER**

Outro método comportamental, que tem como principal objetivo garantir que a instrução fornecida seja reconhecida e executada, independente de possíveis erros de gramática ou lógica.

* Problema: Imagine que você precisa receber um dado de escrita complexa, que tenha caracteres especiais, e/ou que pode ser escrito de maneira diferente de acordo com o contexto, isso aumenta exponencialmente a chance dos erros de sintaxe. Colocando em um exemplo: Você precisa receber alguma data em seu programa, porém, datas são escritas em ordens diferentes de acordo com o país, além de caracteres de separação variáveis, como “/” ou “.”
* Solução: O interpreter permite com que você defina uma gramática para uma linguagem, e então, o dado será analisado pelo contexto passado, se as informações baterem, o interpretador validará o dado e guardará seu resultado.



Concluindo: Embora possa utilizar mais espaço de código e de armazenamento, o Interpreter é fundamental para códigos que tem uma gramática variável e contextualizada, se bem programado, será de grande auxílio na acessibilidade do software.

**REFERÊNCIAS**

< <https://www.opus-software.com.br/insights/design-patterns/>>

<<https://www.freecodecamp.org/portuguese/news/os-3-tipos-de-padroes-de-projetos-que-todo-desenvolvedor-deveria-conhecer-com-exemplos-de-codigo-de-cada-um/#:~:text=O%20que%20%C3%A9%20um%20padr%C3%A3o,problemas%20e%20projetar%20uma%20solu%C3%A7%C3%A3o>.>

< <https://refactoring.guru/pt-br/design-patterns/catalog>>

<<https://www.devmedia.com.br/introducao-aos-padroes-criacionais-abstract-factory-factory-method-prototype-e-singleton/21249>>

<<https://www.treinaweb.com.br/blog/padroes-de-projeto-o-que-sao-e-o-que-resolvem>>

<<https://www.devmedia.com.br/patterns-template-method/18953>>

<<https://medium.com/xp-inc/design-patterns-parte-24-template-method-69e3a7927dcd>>

<<https://medium.com/xp-inc/design-patterns-parte-17-interpreter-dce71780570a>>